**Assignment 18**

**1.What is the difference between a generator and a function?**

Function can be executed only from top to bottom. The only way to exit the function, is by returning from it or throwing an error. If the function is invoked again, it will start the execution from the top to bottom.

A normal function cannot pause until its execution is completed. The only method to break out of a normal function would be to use the return keyword. There are some advantages of using generator function over normal function is Memory-efficient , easier to test and does not calculate a value unless it is needed.

Generator:

It is a new kind of function that is different from a regular function, which can be paused midway and then continues from where it paused. It is denoted by a symbol of asterisk(\*). At first, object is created for generator functions and this objects is iterable . This holds two things: one is value and another one is done. This done value will be false till last value is used..when no other value is yet to process or execute..when next() is called then it will give { value: undefined, done: true }

When such function is called, it doesn't run its code. Instead it returns a special object, called “generator object”, to manage the execution. When next() is called.

A function that generates a series of values instead of a single value and can pause and resume when required during the execution. It includes an operator called yield which allows pausing the generator function itself until the next value is requested.

Generator function is denoted by

**function\* generate () or function \*generate(){ // }**

example:

function\* generate()

{

    console.log("invoked first time")

    yield 1;

    console.log("invoked second time")

    yield 2;

}

let gen=generate(); //object is created for generator function

console.log(gen.next()); //Generator starts by invoking next method.

console.log(gen.next());

console.log(gen.next().done);

*//when next is called first time it will invoke first time stmts and execute till yield 1*

invoked first time

{ value: 1, done: false }

*//when next is called second time it will execute stmts(after yield 1 and execute till yield 2)*

invoked second time

{ value: 2, done: false }

//when next()is called next time,then no stmt to execute so it will give done as true and value as undefined

**2. What is the syntax of a generator?**

function\* functionname ()

{ //code to be executed ;}

let gen=generate(); // here object will be generated.

or

function \*functionname()

{ //code to be executed }

With the help of this object, generator will be executed.

**3. Are function generators iterable in JavaScript?**

Yes, generator function is iterable in javascript..we can use for of loop to iterate .

function\* generate()

{

    console.log("invoked first time")

    yield 1;

    console.log("invoked second time")

    yield 2;

}

let gen=generate();

for(const g of gen)

{

    console.log(g)

}

Instead of using gen.next() repeatedly to execute next steps, we can use above for loop to execute the same.

Output:

invoked first time

1

invoked second time

2

Another example:

function \*generator()

{

    let index=0;

    while(index<3)

    {

        yield index++;

    }

}

let gen=generator();

for(let g of gen)

{

    console.log(g) //using for of loop to iterate .

}

Output:

0

1

2

**4. Create a generator for multiplying?**

function \*multiply()

{

    const number=yield "Give number for multiply"

    yield number\*2;

     const number1=yield "Give number to add"

    yield number1+3;

}

const mul=multiply();

console.log(mul.next().value); // the question is shown as Give number for multiply

console.log("numberis passed as 2 in mul.next(2).value...2\*2=4")

console.log(mul.next(2).value);

console.log(mul.next().value); // the question is shown as Give number to add

console.log("Number is passed in argument in mul.next(3).value..3+3=6")

console.log(mul.next(3).value)

output:

[Running] node "c:\rukshana-proj\blockchain\blockchain\multiplygeneratorfunction.js"

Give number for multiply

numberis passed as 2 in mul.next(2).value...2\*2=4

4

Give number to add

Number is passed in argument in mul.next(3).value..3+3=6

6

Example..if we need to multiply same number… i value can be passed through generators.

function \*generator(i)

{

yield i;

yield i+i;

yield i\*i;

}

const gen=generator(15); //passed i value as 15

console.log(gen.next());

// when next() is executed it will execute i+i;

console.log("--------------------")

console.log(gen.next())

console.log(gen.next())// this will execute i\*i

output:

{ value: 15, done: false }

--------------------

{ value: 30, done: false }

{ value: 225, done: false }

5**. Print an infinite series of natural numbers using a generator**

function \*generator()

{

    let index=1;

    while(true) //while(true)—executes infinite times

    {

        yield index++;

    }

}

let gen=generator();

for(let g of gen)

{

    console.log(g)

}

If we give any condition within a while loop, when the condition fails it will come out of loop..but here to print an infinite series of natural numbers..so index is assigned to 1 …for-of loop is used to iterate the values.

**6. Create a generator that can throw an exception.**

[**Syntax**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Generator/throw#syntax)

generatorObject.throw(exception)

The throw() method of a generator acts as if a throw statement is inserted in the generator body at the current suspended position,which tells the generator about error condition ..whether to handle the error or to stop the process. Throw holds exception as a parameter. For debugging purposes, it is useful to make it an instanceof [Error](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Error).

If in case thrown exception is not caught by a try..catch,then the exception passed to throw(),then from the generator function the exception will be thrown out.

The following example shows a simple generator and an error that is thrown using the throw method. An error can be caught by a [try...catch](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch) block as usual.

function\* gen() {

while (true) {

try {

yield ‘hi’;

} catch (e) {

console.log('oops….Error occurred!');

}

}

}

const g = gen();

g.next();

g.throw(new Error('Something went wrong'));

Output:

oops….Error occurred!

Another example:

In this function, throw()is given in the generator function. i value will be printed using yield i, when it exist the condition of i>5 then only the error is thrown, which is caught by catch block and the error is given as Throwing an error because x value crossed the limit of 5

function \*exceptionFun()

{

for(i=0;i<8;i++)

{

    yield i;

    if(i>=5)

     throw new Error('because x value crossed the limit of 5')

}

}

try{

for(let g of exceptionFun())

console.log(g)

console.log('not throwing an error till I value is 5');  //wont throw till value becomes 5

     }

    catch(e)

    {

console.log("Throwing an ",e)

    }

Output:

0

1

2

3

4

5

Throwing an error because x value crossed the limit of 5

    at exceptionFun (c:\rukshana-proj\js assign\exceptionhandlinggenerators.js:7:12)